Supervised Learning - Foundations Project: ReCell

Problem Statement

Business Context

Buying and selling used phones and tablets used to be something that happened on a handful of online marketplace sites. But the used and refurbished device market has grown considerably over the past decade, and a new IDC (International Data Corporation) forecast predicts that the used phone market would be worth \$52.7bn by 2023 with a compound annual growth rate (CAGR) of 13.6% from 2018 to 2023. This growth can be attributed to an uptick in demand for used phones and tablets that offer considerable savings compared with new models.

Refurbished and used devices continue to provide cost-effective alternatives to both consumers and businesses that are looking to save money when purchasing one. There are plenty of other benefits associated with the used device market. Used and refurbished devices can be sold with warranties and can also be insured with proof of purchase. Third-party vendors/platforms, such as Verizon, Amazon, etc., provide attractive offers to customers for refurbished devices. Maximizing the longevity of devices through second-hand trade also reduces their environmental impact and helps in recycling and reducing waste. The impact of the COVID-19 outbreak may further boost this segment as consumers cut back on discretionary spending and buy phones and tablets only for immediate needs.

Objective

The rising potential of this comparatively under-the-radar market fuels the need for an ML-based solution to develop a dynamic pricing strategy for used and refurbished devices. ReCell, a startup aiming to tap the potential in this market, has hired you as a data scientist. They want you to analyze the data provided and build a linear regression model to predict the price of a used phone/tablet and identify factors that significantly influence it.

Data Description

The data contains the different attributes of used/refurbished phones and tablets. The data was collected in the year 2021. The detailed data dictionary is given below.

- brand_name: Name of manufacturing brand
- os: OS on which the device runs
- screen_size: Size of the screen in cm
- 4g: Whether 4G is available or not
- 5g: Whether 5G is available or not
- main_camera_mp: Resolution of the rear camera in megapixels
- selfie_camera_mp: Resolution of the front camera in megapixels
- int_memory: Amount of internal memory (ROM) in GB
- ram: Amount of RAM in GB
- battery: Energy capacity of the device battery in mAh
- weight: Weight of the device in grams
- release_year: Year when the device model was released
- days_used: Number of days the used/refurbished device has been used
- normalized_new_price: Normalized price of a new device of the same model in euros
- normalized_used_price: Normalized price of the used/refurbished device in euros

Importing necessary libraries

```
# Installing the libraries with the specified version.
# uncomment and run the following line if Google Colab is being used
!pip install scikit-learn==1.2.2 seaborn==0.13.1 matplotlib==3.7.1 numpy==1.25.2 pandas==1.5.3 -q --user
```

Note: *After running the above cell, kindly restart the notebook kernel and run all cells sequentially from the start again.*

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
sns.set()

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

import statsmodels.api as sm

from statsmodels.stats.outliers_influence import variance_inflation_factor
```

Loading the dataset

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/3 - Supervised Learning - Foundations/Final Project/used_device_data.csv')
```

Data Overview

Loading the dataset

```
df.head()
```

	brand_name	os	screen_size	4g	5g	main_camera_mp	selfie_camera_mp	int_memory	ram	battery	weight	release_year	days_used	normalized_used_price	normalized_
0	Honor	Android	14.50	yes	no	13.0	5.0	64.0	3.0	3020.0	146.0	2020	127	4.307572	
1	Honor	Android	17.30	yes	yes	13.0	16.0	128.0	8.0	4300.0	213.0	2020	325	5.162097	
2	Honor	Android	16.69	yes	yes	13.0	8.0	128.0	8.0	4200.0	213.0	2020	162	5.111084	
3	Honor	Android	25.50	yes	yes	13.0	8.0	64.0	6.0	7250.0	480.0	2020	345	5.135387	
4	Honor	Android	15.32	yes	no	13.0	8.0	64.0	3.0	5000.0	185.0	2020	293	4.389995	

Next steps: Generate code with df | View recommended plots | New interactive sheet

Shape of the dataset

```
df.shape
```

```
(3454, 15)
```

Observations - There are 3,454 rows and 15 columns in the dataset

Info regarding column datatypes

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3454 entries, 0 to 3453
Data columns (total 15 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   brand_name             3454 non-null   object 
 1   os                     3454 non-null   object 
 2   screen_size            3454 non-null   float64
 3   4g                     3454 non-null   object 
 4   5g                     3454 non-null   object 
 5   main_camera_mp         3275 non-null   float64
 6   selfie_camera_mp       3452 non-null   float64
 7   int_memory             3450 non-null   float64
 8   ram                    3450 non-null   float64
 9   battery                3448 non-null   float64
 10  weight                 3447 non-null   float64
 11  release_year           3454 non-null   int64  
 12  days_used              3454 non-null   int64  
 13  normalized_used_price  3454 non-null   float64
 14  normalized_new_price   3454 non-null   float64
dtypes: float64(9), int64(2), object(4)
memory usage: 404.9+ KB
```

Observations - There are 11 numerical (2 int64 & 9 float64) and 4 object type columns in the dataset

Statistics summary for the numerical columns

```
df.describe()
```

	screen_size	main_camera_mp	selfie_camera_mp	int_memory	ram	battery	weight	release_year	days_used	normalized_used_price	normalized_new_pri
count	3454.000000	3275.000000	3452.000000	3450.000000	3450.000000	3448.000000	3447.000000	3454.000000	3454.000000	3454.000000	3454.000
mean	13.713115	9.460208	6.554229	54.573099	4.036122	3133.402697	182.751871	2015.965258	674.869716	4.364712	5.2331
std	3.805280	4.815461	6.970372	84.972371	1.365105	1299.682844	88.413228	2.298455	248.580166	0.588914	0.6836
min	5.080000	0.080000	0.000000	0.010000	0.020000	500.000000	69.000000	2013.000000	91.000000	1.536867	2.9014
25%	12.700000	5.000000	2.000000	16.000000	4.000000	2100.000000	142.000000	2014.000000	533.500000	4.033931	4.7903
50%	12.830000	8.000000	5.000000	32.000000	4.000000	3000.000000	160.000000	2015.500000	690.500000	4.405133	5.2458
75%	15.340000	13.000000	8.000000	64.000000	4.000000	4000.000000	185.000000	2018.000000	868.750000	4.755700	5.6737
max	30.710000	48.000000	32.000000	1024.000000	12.000000	9720.000000	855.000000	2020.000000	1094.000000	6.619433	7.8478

Checking missing values

```
df.isnull().sum()
```

	0
brand_name	0
os	0
screen_size	0
4g	0
5g	0
main_camera_mp	179
selfie_camera_mp	2
int_memory	4
ram	4
battery	6
weight	7
release_year	0
days_used	0
normalized_used_price	0
normalized_new_price	0

dtype: int64

Observations - There are missing values in the following columns:

- *main_camera_mp*
- *selfie_camera_mp*
- *int_memory*
- *ram*
- *battery*
- *weight*

Check for duplicates in the dataset

```
print("There are",df.duplicated().sum(),"duplicated rows")
```

```
There are 0 duplicated rows
```

Exploratory Data Analysis (EDA)

- EDA is an important part of any project involving data.
- It is important to investigate and understand the data better before building a model with it.
- A few questions have been mentioned below which will help you approach the analysis in the right manner and generate insights from the data.
- A thorough analysis of the data, in addition to the questions mentioned below, should be done.

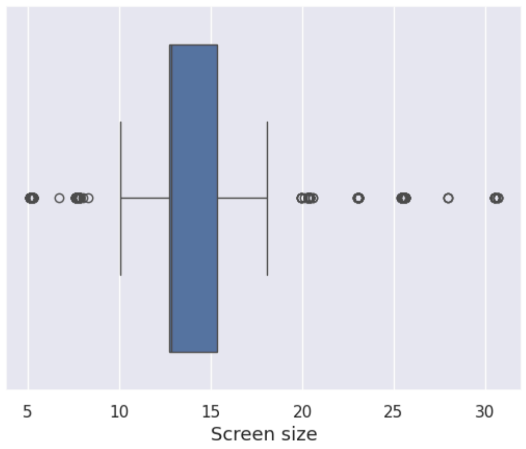
Questions:

1. What does the distribution of normalized used device prices look like?
2. What percentage of the used device market is dominated by Android devices?
3. The amount of RAM is important for the smooth functioning of a device. How does the amount of RAM vary with the brand?
4. A large battery often increases a device's weight, making it feel uncomfortable in the hands. How does the weight vary for phones and tablets offering large batteries (more than 4500 mAh)?
5. Bigger screens are desirable for entertainment purposes as they offer a better viewing experience. How many phones and tablets are available across different brands with a screen size larger than 6 inches?
6. A lot of devices nowadays offer great selfie cameras, allowing us to capture our favorite moments with loved ones. What is the distribution of devices offering greater than 8MP selfie cameras across brands?
7. Which attributes are highly correlated with the normalized price of a used device?

Univariate Analysis

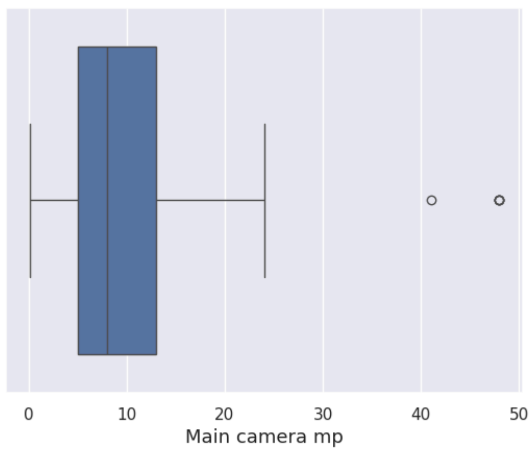
screen_size

```
chart = sns.boxplot(data=df,x='screen_size')
chart.set_xlabel('Screen size', fontdict={'size': 13})
plt.show()
```



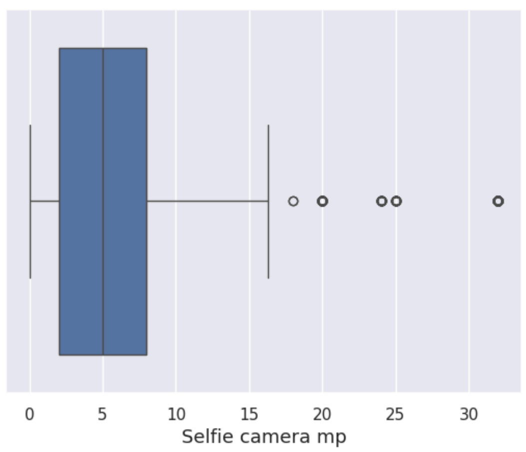
main_camera_mp

```
chart = sns.boxplot(data=df,x='main_camera_mp')
chart.set_xlabel('Main camera mp', fontdict={'size': 13})
plt.show()
```



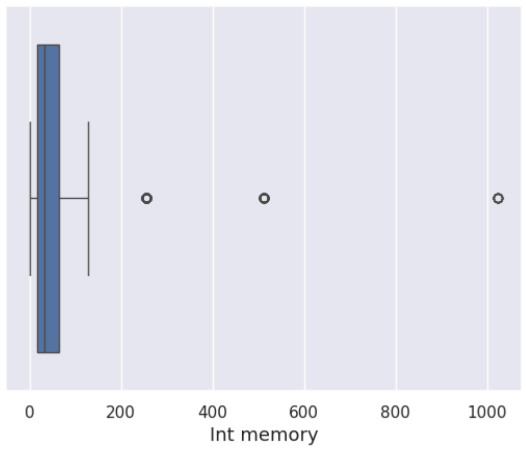
selfie_camera_mp

```
chart = sns.boxplot(data=df,x='selfie_camera_mp')
chart.set_xlabel('Selfie camera mp', fontdict={'size': 13})
plt.show()
```



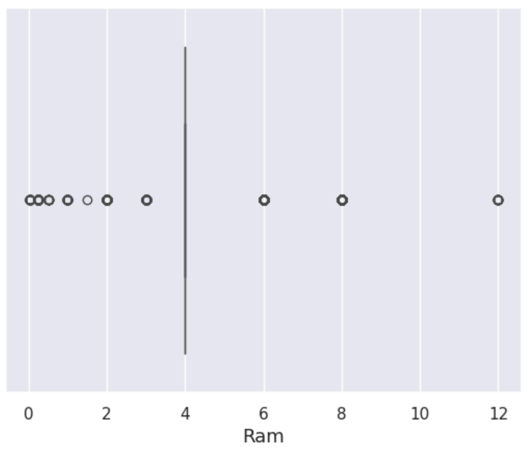
int_memory

```
chart = sns.boxplot(data=df,x='int_memory')
chart.set_xlabel('Int memory', fontdict={'size': 13})
plt.show()
```



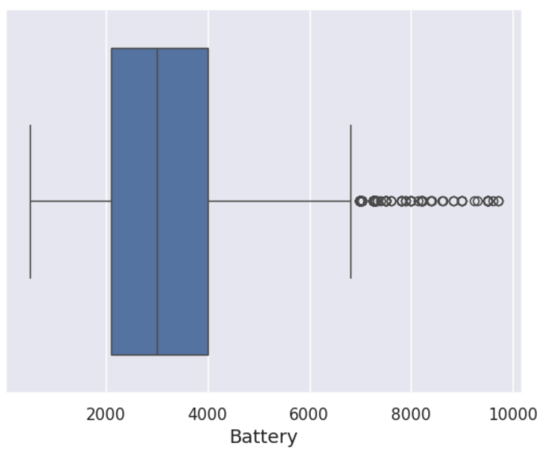
ram

```
chart = sns.boxplot(data=df,x='ram')
chart.set_xlabel('Ram', fontdict={'size': 13})
plt.show()
```



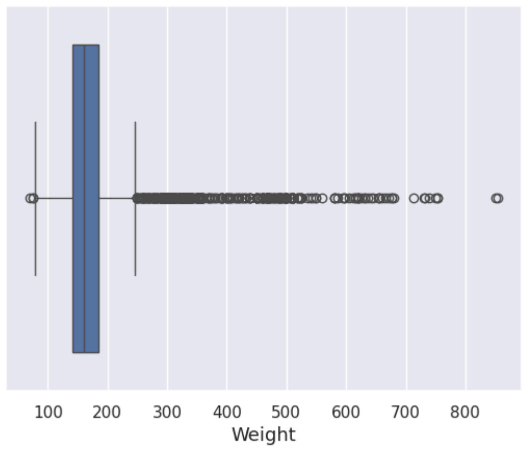
battery

```
chart = sns.boxplot(data=df,x='battery')
chart.set_xlabel('Battery', fontdict={'size': 13})
plt.show()
```



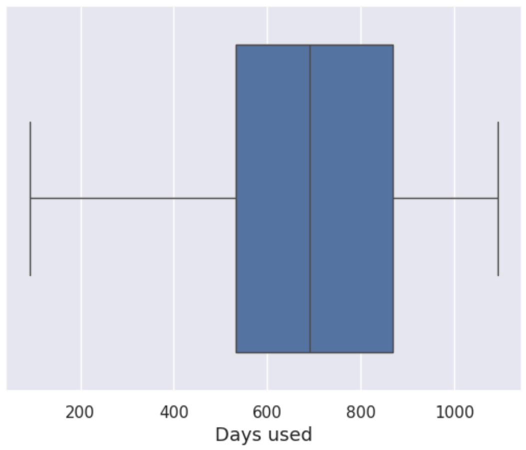
weight

```
chart = sns.boxplot(data=df,x='weight')
chart.set_xlabel('Weight', fontdict={'size': 13})
plt.show()
```



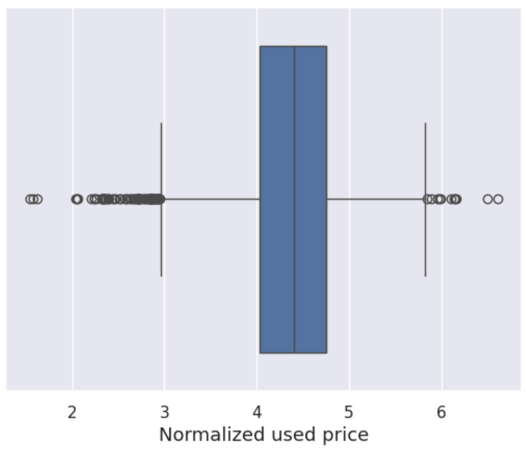
days_used

```
chart = sns.boxplot(data=df,x='days_used')
chart.set_xlabel('Days used', fontdict={'size': 13})
plt.show()
```



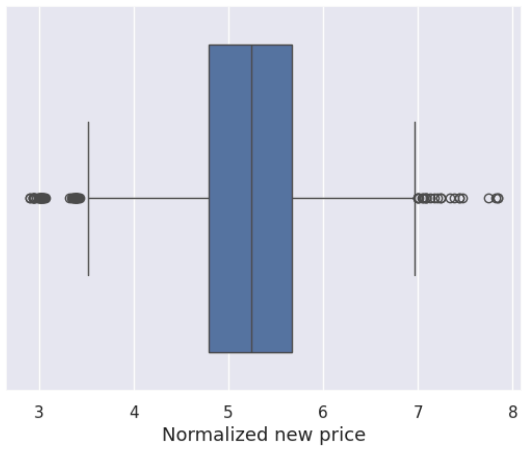
normalized_used_price

```
chart = sns.boxplot(data=df,x='normalized_used_price')
chart.set_xlabel('Normalized used price', fontdict={'size': 13})
plt.show()
```



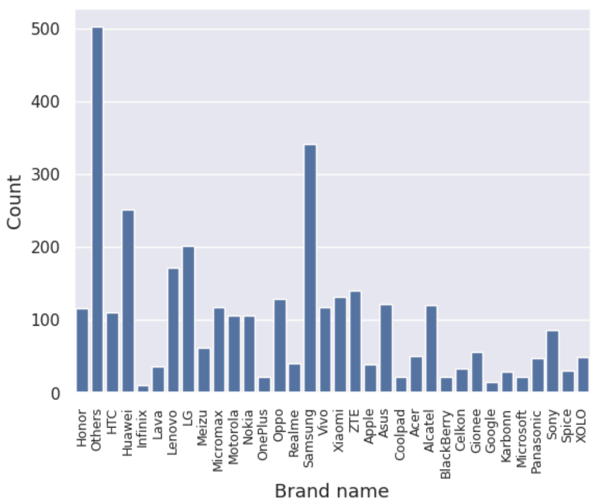
normalized_new_price

```
chart = sns.boxplot(data=df,x='normalized_new_price')
chart.set_xlabel('Normalized new price', fontdict={'size': 13})
plt.show()
```



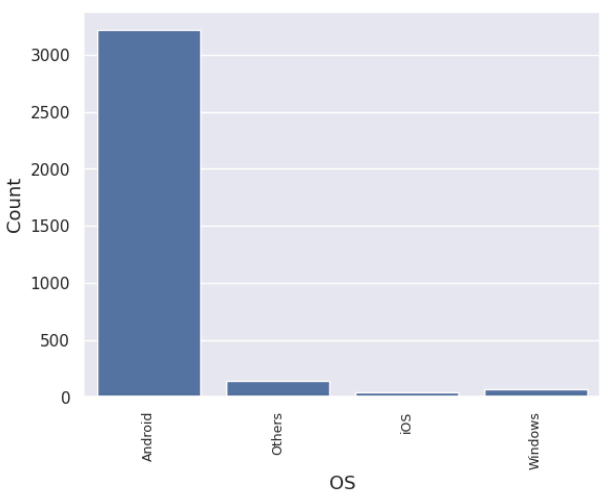
brand_name

```
chart = sns.countplot(data=df,x='brand_name')
chart.set_xlabel('Brand name', fontdict={'size': 13})
chart.set_ylabel('Count', fontdict={'size': 13})
plt.xticks(rotation=90,fontsize=9)
plt.show()
```



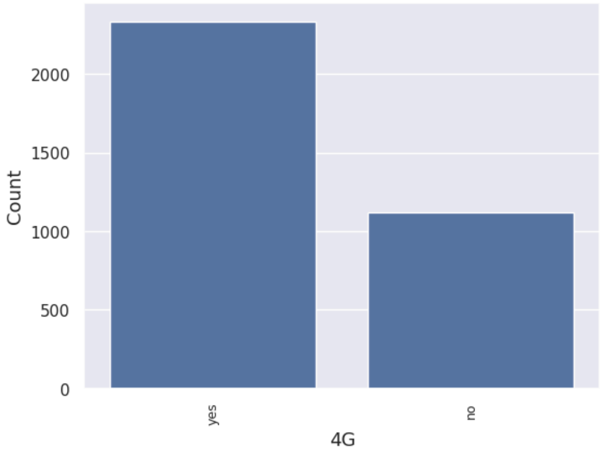
os

```
chart = sns.countplot(data=df,x='os')
chart.set_xlabel('OS', fontdict={'size': 13})
chart.set_ylabel('Count', fontdict={'size': 13})
plt.xticks(rotation=90,fontsize=9)
plt.show()
```



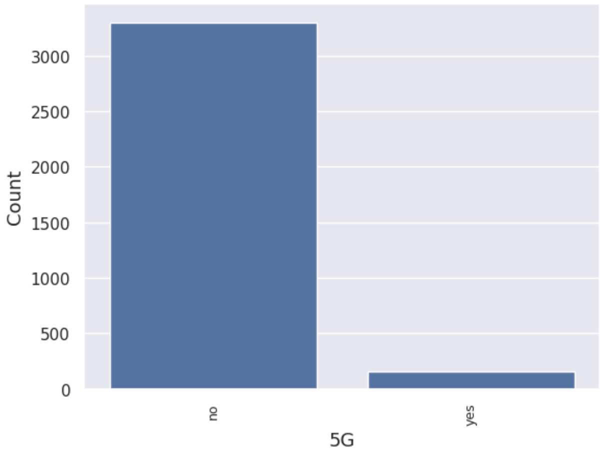
4g

```
chart = sns.countplot(data=df,x='4g')
chart.set_xlabel('4G', fontdict={'size': 13})
chart.set_ylabel('Count', fontdict={'size': 13})
plt.xticks(rotation=90,fontsize=9)
plt.show()
```



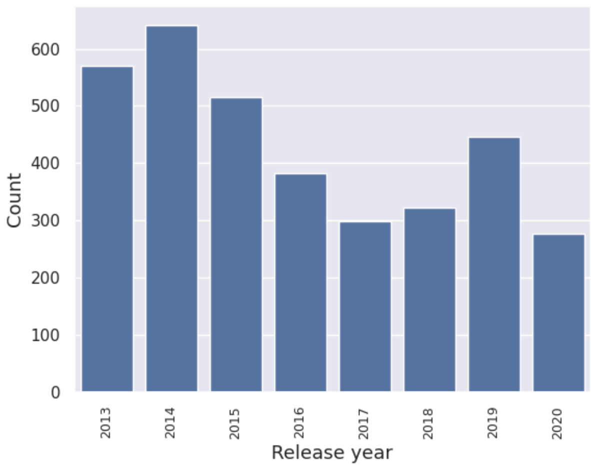
5g

```
chart = sns.countplot(data=df,x='5g')
chart.set_xlabel('5G', fontdict={'size': 13})
chart.set_ylabel('Count', fontdict={'size': 13})
plt.xticks(rotation=90,fontsize=9)
plt.show()
```



release_year

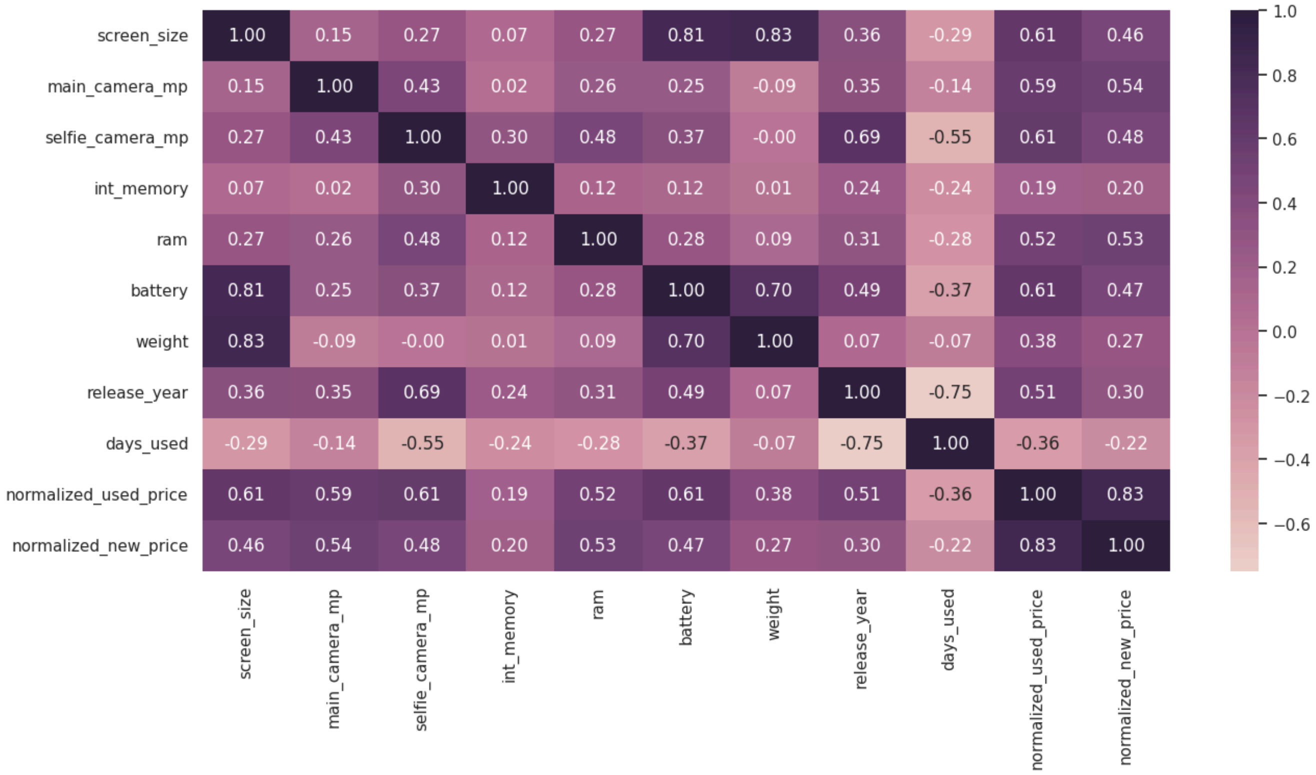
```
chart = sns.countplot(data=df,x='release_year')
chart.set_xlabel('Release year', fontdict={'size': 13})
chart.set_ylabel('Count', fontdict={'size': 13})
plt.xticks(rotation=90,fontsize=9)
plt.show()
```



Bivariate Analysis

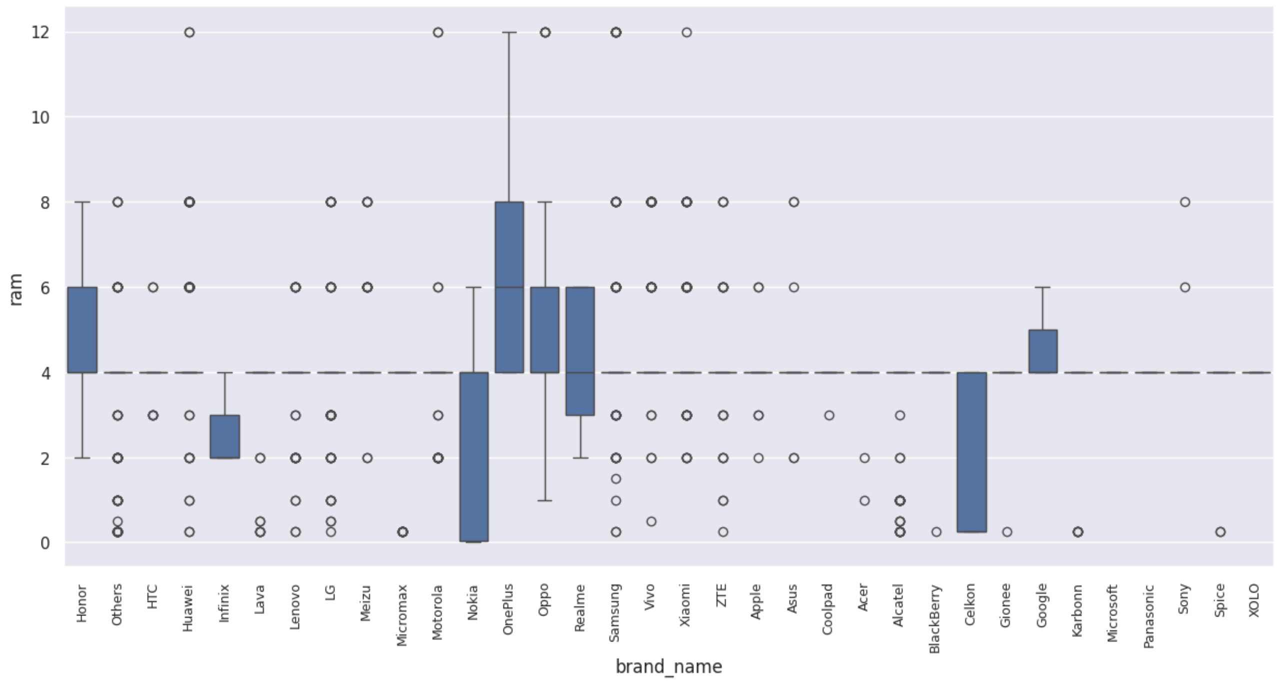
Correlation analysis

```
data = df.select_dtypes(include=np.number).columns.tolist()

plt.figure(figsize=(15, 7))
sns.heatmap(df[data].corr(), annot=True, fmt=".2f", cmap=sns.cubehelix_palette(as_cmap=True))
plt.show()
```



Brand name vs ram

```
plt.figure(figsize=(15, 7))

sns.boxplot(data=df, x="brand_name", y="ram")
plt.xticks(rotation=90,fontsize=9)
plt.show()
```

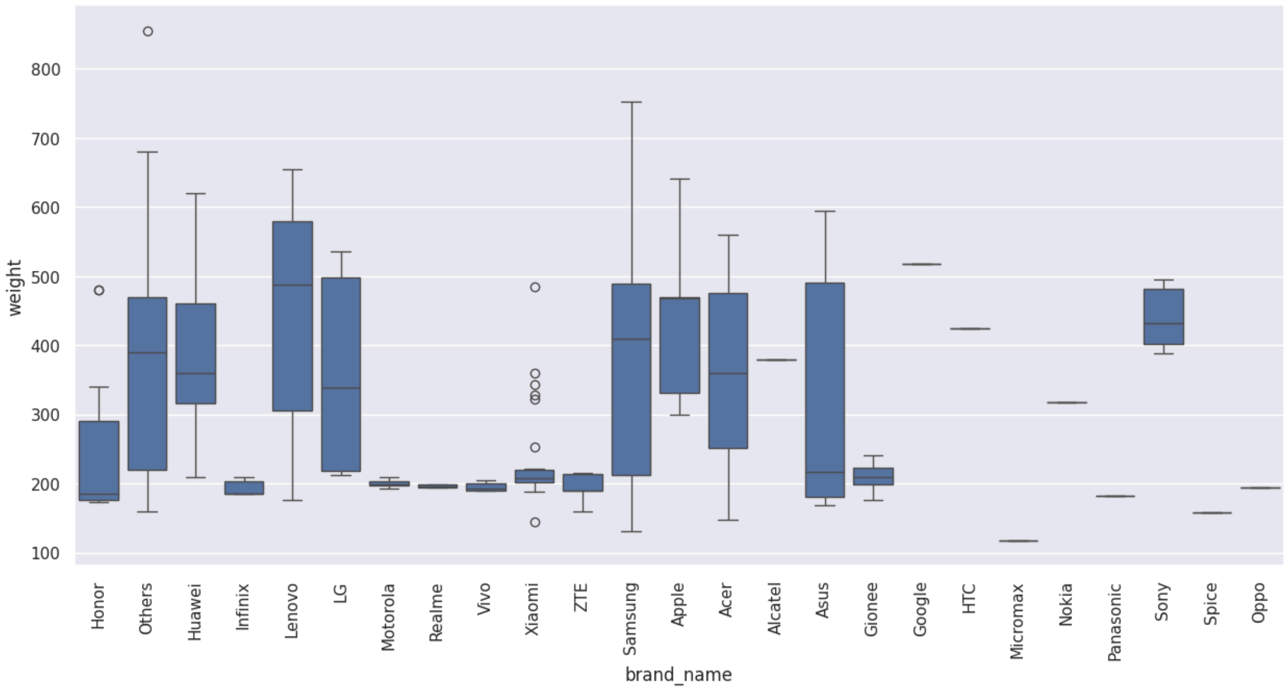


Large battery analysis

```
data = df.loc[df.battery > 4500]
data.shape
```

```
(341, 15)
```

```
plt.figure(figsize=(15, 7))
sns.boxplot(data=data, x="brand_name", y="weight")
plt.xticks(rotation=90)
plt.show()
```

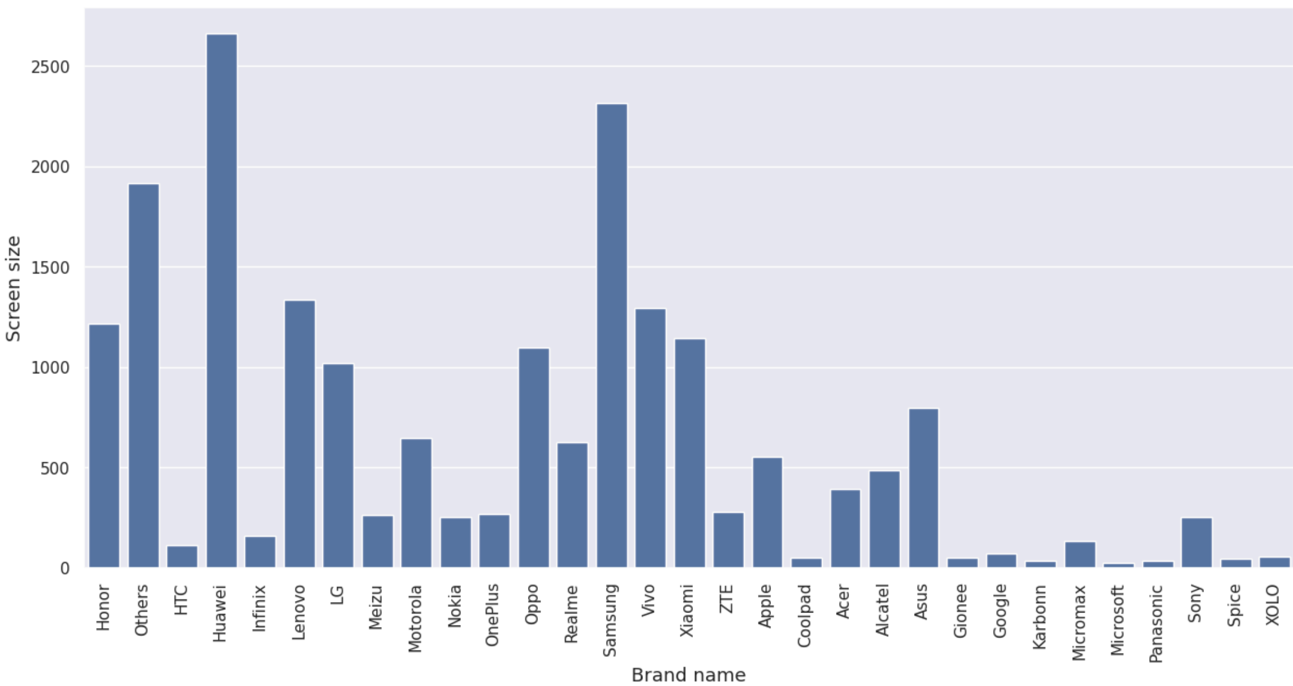


Screens analysis

```
data = df.loc[df.screen_size > (6 * 2.54)] # 1 inch equals 2.54 cm
data.shape
```

```
(1099, 15)
```

```
plt.figure(figsize=(15, 7))
chart = sns.barplot(data=data,x='brand_name',y='screen_size',estimator="sum",errorbar=None)
chart.set_xlabel('Brand name', fontdict={'size': 13})
chart.set_ylabel('Screen size', fontdict={'size': 13})
plt.xticks(rotation=90)
plt.show()
```

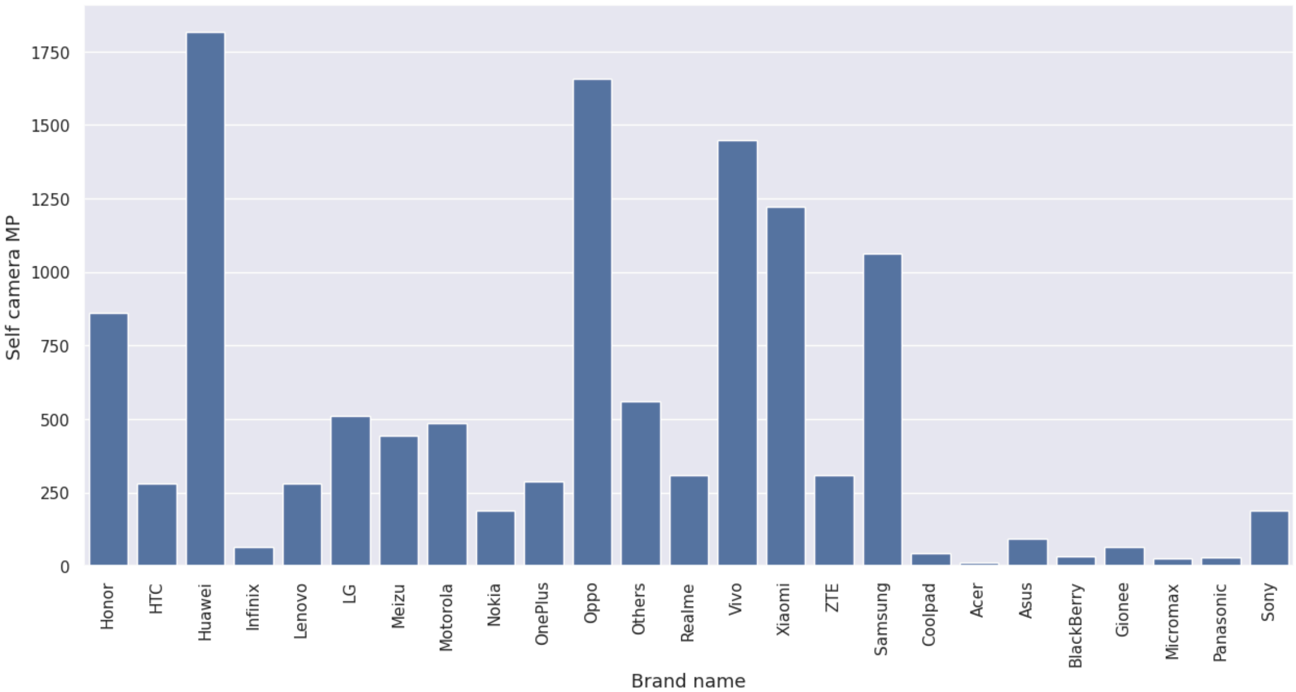


Selfie cameras analysis

```
data = df.loc[df.selfie_camera_mp > 8]
data.shape
```

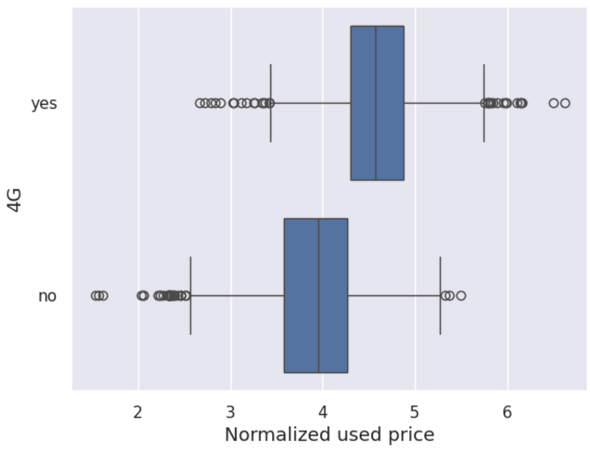
```
(655, 15)
```

```
plt.figure(figsize=(15, 7))
chart = sns.barplot(data=data,x='brand_name',y='selfie_camera_mp',estimator="sum",errorbar=None)
chart.set_xlabel('Brand name', fontdict={'size': 13})
chart.set_ylabel('Self camera MP', fontdict={'size': 13})
plt.xticks(rotation=90)
plt.show()
```



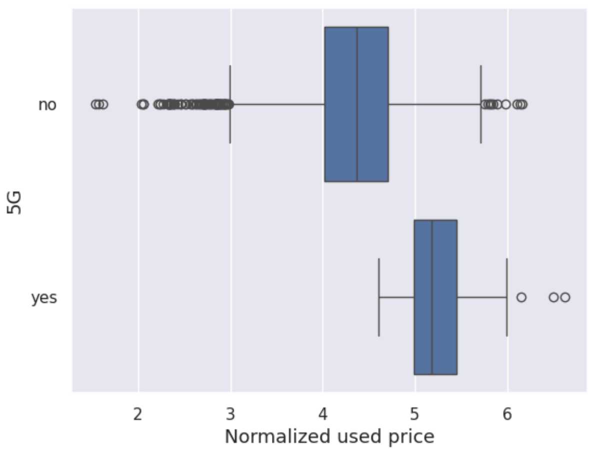
4G vs Normalized used price

```
chart = sns.boxplot(data=df,x='normalized_used_price',y='4g')
chart.set_xlabel('Normalized used price', fontdict={'size': 13})
chart.set_ylabel('4G', fontdict={'size': 13})
plt.show()
```



5G vs Normalized used price

```
chart = sns.boxplot(data=df,x='normalized_used_price',y='5g')
chart.set_xlabel('Normalized used price', fontdict={'size': 13})
chart.set_ylabel('5G', fontdict={'size': 13})
plt.show()
```



Data Preprocessing

- Missing value treatment
- Feature engineering (if needed)
- Outlier detection and treatment (if needed)
- Preparing data for modeling
- Any other preprocessing steps (if needed)

Copy data

```
dfCopy = df.copy()
```

Missing value treatment

```
dfCopy.isnull().sum()
```

	0
brand_name	0
os	0
screen_size	0
4g	0
5g	0
main_camera_mp	179
selfie_camera_mp	2
int_memory	4
ram	4
battery	6
weight	7
release_year	0
days_used	0
normalized_used_price	0
normalized_new_price	0

dtype: int64

```
columns = ["main_camera_mp","selfie_camera_mp","int_memory","ram","battery","weight"]

for column in columns:
    dfCopy[column] = dfCopy[column].fillna(
      value=dfCopy.groupby(["brand_name"])[column].transform("median")
    )

# re-check for missing values
dfCopy.isnull().sum()
```

	0
brand_name	0
os	0
screen_size	0
4g	0
5g	0
main_camera_mp	10
selfie_camera_mp	0
int_memory	0
ram	0
battery	0
weight	0
release_year	0
days_used	0
normalized_used_price	0
normalized_new_price	0

dtype: int64

```
dfCopy["main_camera_mp"] = dfCopy["main_camera_mp"].fillna(value=dfCopy["main_camera_mp"].median())

# re-check for missing values
dfCopy.isnull().sum()
```

	0
brand_name	0
os	0
screen_size	0
4g	0
5g	0
main_camera_mp	0
selfie_camera_mp	0
int_memory	0
ram	0
battery	0
weight	0
release_year	0
days_used	0
normalized_used_price	0
normalized_new_price	0

dtype: int64

Feature engineering

- Dropping release_year and creating release_age column

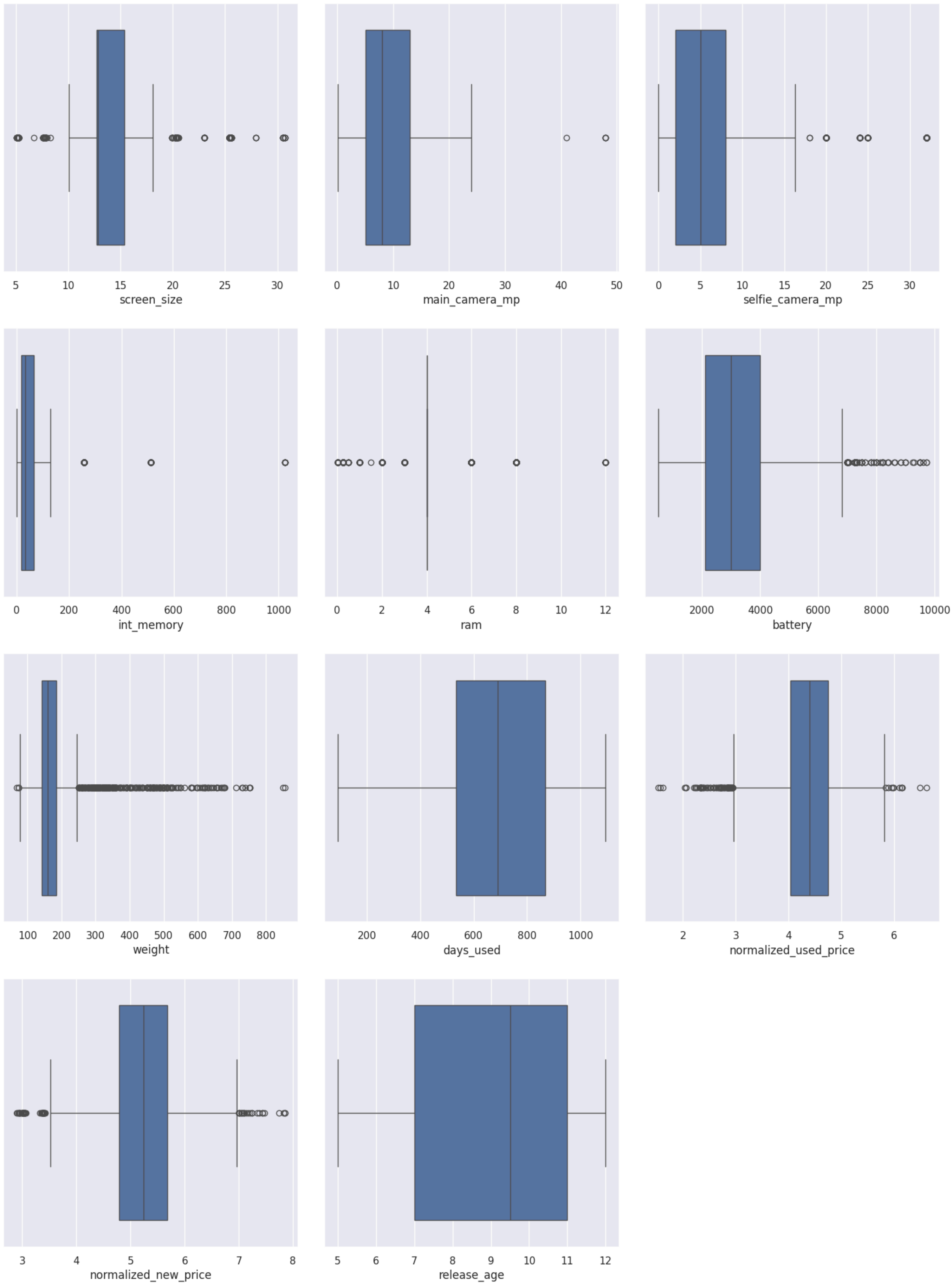
```
from datetime import datetime

dfCopy["release_age"] = datetime.now().date().year - dfCopy["release_year"]
dfCopy.drop("release_year", axis=1, inplace=True)
dfCopy["release_age"].describe()
```

	release_age
count	3454.000000
mean	9.034742
std	2.298455
min	5.000000
25%	7.000000
50%	9.500000
75%	11.000000
max	12.000000

dtype: float64

Outlier detection and treatment

```
num_cols = dfCopy.select_dtypes(include=np.number).columns.tolist()
num_rows = (len(num_cols) + 2) // 3
num_cols_per_row = 3

plt.figure(figsize=(15, 5 * num_rows))

for i, variable in enumerate(num_cols):
    plt.subplot(num_rows, num_cols_per_row, i + 1)
    sns.boxplot(data=dfCopy, x=variable)
    plt.tight_layout(pad=2)

plt.show()
```


screen_size
main_camera_mp
selfie_camera_mp
int_memory
ram
battery
weight
days_used
normalized_used_price
normalized_new_price
release_age

Preparing data for modeling

- as per objective, we want to predict the price for used devices

```
# splitting the data into the dependent and independent variables
X = dfCopy.drop(columns=['normalized_used_price'], axis=1)
y = dfCopy['normalized_used_price']

print(X.head())
print(y.head())
```

```
  brand_name       os  screen_size   4g   5g  main_camera_mp  \
0      Honor  Android        14.50  yes   no            13.0
1      Honor  Android        17.30  yes  yes            13.0
2      Honor  Android        16.69  yes  yes            13.0
3      Honor  Android        25.50  yes  yes            13.0
4      Honor  Android        15.32  yes   no            13.0

   selfie_camera_mp  int_memory  ram  battery  weight  days_used  \
0               5.0        64.0  3.0   3020.0   146.0        127
1              16.0       128.0  8.0   4300.0   213.0        325
2               8.0       128.0  8.0   4200.0   213.0        162
3               8.0        64.0  6.0   7250.0   480.0        345
4               8.0        64.0  3.0   5000.0   185.0        293

   normalized_new_price  release_age
0              4.715100            5
1              5.519018            5
2              5.884631            5
3              5.630961            5
4              4.947837            5
0    4.307572
1    5.162097
2    5.111084
3    5.135387
4    4.389995
Name: normalized_used_price, dtype: float64
```

```
# let's add the intercept to data
X = sm.add_constant(X)
```

```
# creating dummy variables
X = pd.get_dummies(
    X,
    columns=X.select_dtypes(include=["object", "category"]).columns.tolist(),
    drop_first=True,
)

# converting the input attributes into float type for modeling
X = X.astype(float)
X.head()
```

	const	screen_size	main_camera_mp	selfie_camera_mp	int_memory	ram	battery	weight	days_used	normalized_new_price	...	brand_name_Spice	brand_name_Vivo	brand_na
0	1.0	14.50	13.0	5.0	64.0	3.0	3020.0	146.0	127.0	4.715100	...	0.0	0.0	
1	1.0	17.30	13.0	16.0	128.0	8.0	4300.0	213.0	325.0	5.519018	...	0.0	0.0	
2	1.0	16.69	13.0	8.0	128.0	8.0	4200.0	213.0	162.0	5.884631	...	0.0	0.0	
3	1.0	25.50	13.0	8.0	64.0	6.0	7250.0	480.0	345.0	5.630961	...	0.0	0.0	
4	1.0	15.32	13.0	8.0	64.0	3.0	5000.0	185.0	293.0	4.947837	...	0.0	0.0	

5 rows × 49 columns

```
# splitting the data in 70:30 ratio for train to test data
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=1)

print("Train data ->", x_train.shape[0],"rows")
print("Test data ->", x_test.shape[0],"rows")
```

```
Train data -> 2417 rows
Test data -> 1037 rows
```

EDA

- It is a good idea to explore the data once again after manipulating it.

```
data = pd.concat([X, y], axis=1)
```

```
# data shape
data.shape
```

```
(3454, 50)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3454 entries, 0 to 3453
Data columns (total 50 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   const                  3454 non-null   float64
 1   screen_size            3454 non-null   float64
 2   main_camera_mp         3454 non-null   float64
 3   selfie_camera_mp       3454 non-null   float64
 4   int_memory             3454 non-null   float64
 5   ram                    3454 non-null   float64
 6   battery                3454 non-null   float64
 7   weight                 3454 non-null   float64
 8   days_used              3454 non-null   float64
```

```
 9   normalized_new_price   3454 non-null   float64
 10  release_age            3454 non-null   float64
 11  brand_name_Alcatel     3454 non-null   float64
 12  brand_name_Apple       3454 non-null   float64
 13  brand_name_Asus        3454 non-null   float64
 14  brand_name_BlackBerry  3454 non-null   float64
 15  brand_name_Celkon      3454 non-null   float64
 16  brand_name_Coolpad     3454 non-null   float64
 17  brand_name_Gionee      3454 non-null   float64
 18  brand_name_Google      3454 non-null   float64
 19  brand_name_HTC         3454 non-null   float64
 20  brand_name_Honor       3454 non-null   float64
 21  brand_name_Huawei      3454 non-null   float64
 22  brand_name_Infinix     3454 non-null   float64
 23  brand_name_Karbonn     3454 non-null   float64
 24  brand_name_LG          3454 non-null   float64
 25  brand_name_Lava        3454 non-null   float64
 26  brand_name_Lenovo      3454 non-null   float64
 27  brand_name_Meizu       3454 non-null   float64
 28  brand_name_Micromax    3454 non-null   float64
 29  brand_name_Microsoft   3454 non-null   float64
 30  brand_name_Motorola    3454 non-null   float64
 31  brand_name_Nokia       3454 non-null   float64
 32  brand_name_OnePlus     3454 non-null   float64
 33  brand_name_Oppo        3454 non-null   float64
 34  brand_name_Others      3454 non-null   float64
 35  brand_name_Panasonic   3454 non-null   float64
 36  brand_name_Realme      3454 non-null   float64
 37  brand_name_Samsung     3454 non-null   float64
 38  brand_name_Sony        3454 non-null   float64
 39  brand_name_Spice       3454 non-null   float64
 40  brand_name_Vivo        3454 non-null   float64
 41  brand_name_XOLO        3454 non-null   float64
 42  brand_name_Xiaomi      3454 non-null   float64
 43  brand_name_ZTE         3454 non-null   float64
 44  os_Others              3454 non-null   float64
 45  os_Windows             3454 non-null   float64
 46  os_iOS                 3454 non-null   float64
 47  4g_yes                 3454 non-null   float64
 48  5g_yes                 3454 non-null   float64
 49  normalized_used_price  3454 non-null   float64
dtypes: float64(50)
memory usage: 1.3 MB
```

```
# data head
data.head()
```

	const	screen_size	main_camera_mp	selfie_camera_mp	int_memory	ram	battery	weight	days_used	normalized_new_price	...	brand_name_Vivo	brand_name_XOLO	brand_nam
0	1.0	14.50	13.0	5.0	64.0	3.0	3020.0	146.0	127.0	4.715100	...	0.0	0.0	
1	1.0	17.30	13.0	16.0	128.0	8.0	4300.0	213.0	325.0	5.519018	...	0.0	0.0	
2	1.0	16.69	13.0	8.0	128.0	8.0	4200.0	213.0	162.0	5.884631	...	0.0	0.0	
3	1.0	25.50	13.0	8.0	64.0	6.0	7250.0	480.0	345.0	5.630961	...	0.0	0.0	
4	1.0	15.32	13.0	8.0	64.0	3.0	5000.0	185.0	293.0	4.947837	...	0.0	0.0	

5 rows × 50 columns

```
# summary statistics
data.describe()
```

	const	screen_size	main_camera_mp	selfie_camera_mp	int_memory	ram	battery	weight	days_used	normalized_new_price	...	brand_name_Vivo	brar
count	3454.0	3454.000000	3454.000000	3454.000000	3454.000000	3454.000000	3454.000000	3454.000000	3454.000000	3454.000000	...	3454.000000	
mean	1.0	13.713115	9.617597	6.555067	54.528428	4.036080	3132.577446	182.636856	674.869716	5.233107	...	0.033874	
std	0.0	3.805280	4.749438	6.968440	84.933275	1.364314	1298.884193	88.360445	248.580166	0.683637	...	0.180930	
min	1.0	5.080000	0.080000	0.000000	0.010000	0.020000	500.000000	69.000000	91.000000	2.901422	...	0.000000	
25%	1.0	12.700000	5.000000	2.000000	16.000000	4.000000	2100.000000	142.000000	533.500000	4.790342	...	0.000000	
50%	1.0	12.830000	8.000000	5.000000	32.000000	4.000000	3000.000000	160.000000	690.500000	5.245892	...	0.000000	
75%	1.0	15.340000	13.000000	8.000000	64.000000	4.000000	4000.000000	185.000000	868.750000	5.673718	...	0.000000	
max	1.0	30.710000	48.000000	32.000000	1024.000000	12.000000	9720.000000	855.000000	1094.000000	7.847841	...	1.000000	

8 rows × 50 columns

```
# check for missing values
data.isnull().sum()
```

	0
const	0
screen_size	0
main_camera_mp	0
selfie_camera_mp	0
int_memory	0
ram	0
battery	0
weight	0
days_used	0
normalized_new_price	0
release_age	0
brand_name_Alcatel	0
brand_name_Apple	0
brand_name_Asus	0
brand_name_BlackBerry	0
brand_name_Celkon	0
brand_name_Coolpad	0
brand_name_Gionee	0
brand_name_Google	0
brand_name_HTC	0
brand_name_Honor	0
brand_name_Huawei	0
brand_name_Infinix	0
brand_name_Karbonn	0
brand_name_LG	0
brand_name_Lava	0
brand_name_Lenovo	0
brand_name_Meizu	0
brand_name_Micromax	0
brand_name_Microsoft	0
brand_name_Motorola	0
brand_name_Nokia	0
brand_name_OnePlus	0
brand_name_Oppo	0
brand_name_Others	0
brand_name_Panasonic	0
brand_name_Realme	0
brand_name_Samsung	0
brand_name_Sony	0
brand_name_Spice	0
brand_name_Vivo	0
brand_name_XOLO	0
brand_name_Xiaomi	0
brand_name_ZTE	0
os_Others	0
os_Windows	0
os_iOS	0
4g_yes	0
5g_yes	0
normalized_used_price	0

dtype: int64

Model Building - Linear Regression

```
x_train
```

	const	screen_size	main_camera_mp	selfie_camera_mp	int_memory	ram	battery	weight	days_used	normalized_new_price	...	brand_name_Spice	brand_name_Vivo	brand_
3026	1.0	10.29	8.0	0.3	16.0	4.0	1800.0	120.0	819.0	4.796204	...	0.0	0.0	
1525	1.0	15.34	13.0	5.0	32.0	4.0	4050.0	225.0	585.0	5.434595	...	0.0	0.0	
1128	1.0	12.70	13.0	5.0	32.0	4.0	2550.0	162.0	727.0	5.137914	...	0.0	0.0	
3003	1.0	12.83	8.0	5.0	16.0	4.0	3200.0	126.0	800.0	5.189228	...	0.0	0.0	
2907	1.0	12.88	13.0	16.0	16.0	4.0	2900.0	160.0	560.0	5.016220	...	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2763	1.0	10.29	8.0	2.0	16.0	4.0	2100.0	155.0	802.0	5.006694	...	1.0	0.0	
905	1.0	10.29	5.0	0.3	16.0	4.0	1800.0	145.0	850.0	5.195454	...	0.0	0.0	
1096	1.0	15.77	13.0	24.0	64.0	4.0	3400.0	162.0	720.0	5.345392	...	0.0	0.0	
235	1.0	15.90	13.0	32.0	128.0	6.0	3750.0	172.0	311.0	5.515845	...	0.0	0.0	
1061	1.0	12.70	13.0	5.0	16.0	4.0	2300.0	133.0	699.0	5.602635	...	0.0	0.0	

2417 rows × 49 columns

```
olsmodel1 = sm.OLS(y_train, x_train).fit()
print(olsmodel1.summary())
```

```
main_camera_mp            0.0208      0.002     13.848      0.000       0.018       0.024
selfie_camera_mp          0.0135      0.001     11.996      0.000       0.011       0.016
int_memory                0.0001   6.97e-05      1.664      0.096   -2.07e-05       0.000
ram                       0.0232      0.005      4.515      0.000       0.013       0.033
battery               -1.686e-05   7.27e-06     -2.318      0.021   -3.11e-05    -2.6e-06
weight                    0.0010      0.000      7.488      0.000       0.001       0.001
days_used              4.196e-05   3.09e-05      1.360      0.174   -1.85e-05       0.000
normalized_new_price      0.4309      0.012     35.134      0.000       0.407       0.455
release_age              -0.0236      0.005     -5.189      0.000      -0.033      -0.015
brand_name_Alcatel        0.0154      0.048      0.324      0.746      -0.078       0.109
brand_name_Apple         -0.0032      0.147     -0.021      0.983      -0.292       0.285
brand_name_Asus           0.0150      0.048      0.313      0.754      -0.079       0.109
brand_name_BlackBerry    -0.0297      0.070     -0.423      0.672      -0.167       0.108
brand_name_Celkon        -0.0463      0.066     -0.699      0.484      -0.176       0.084
brand_name_Coolpad        0.0209      0.073      0.286      0.775      -0.122       0.164
brand_name_Gionee         0.0447      0.058      0.775      0.438      -0.068       0.158
brand_name_Google        -0.0327      0.085     -0.386      0.700      -0.199       0.133
brand_name_HTC           -0.0131      0.048     -0.271      0.786      -0.108       0.081
brand_name_Honor          0.0316      0.049      0.642      0.521      -0.065       0.128
brand_name_Huawei        -0.0022      0.044     -0.049      0.961      -0.089       0.085
brand_name_Infinix        0.1634      0.093      1.753      0.080      -0.019       0.346
brand_name_Karbonn        0.0943      0.067      1.406      0.160      -0.037       0.226
brand_name_LG            -0.0132      0.045     -0.292      0.771      -0.102       0.076
brand_name_Lava           0.0332      0.062      0.533      0.594      -0.089       0.155
brand_name_Lenovo         0.0453      0.045      1.003      0.316      -0.043       0.134
brand_name_Meizu         -0.0130      0.056     -0.232      0.817      -0.123       0.097
brand_name_Micromax      -0.0337      0.048     -0.704      0.481      -0.128       0.060
brand_name_Microsoft      0.0947      0.088      1.072      0.284      -0.079       0.268
brand_name_Motorola      -0.0113      0.050     -0.228      0.820      -0.109       0.086
brand_name_Nokia          0.0705      0.052      1.362      0.173      -0.031       0.172
brand_name_OnePlus        0.0707      0.077      0.913      0.361      -0.081       0.222
brand_name_Oppo           0.0124      0.048      0.259      0.796      -0.081       0.106
brand_name_Others        -0.0080      0.042     -0.191      0.849      -0.091       0.074
brand_name_Panasonic      0.0562      0.056      1.006      0.314      -0.053       0.166
brand_name_Realme         0.0319      0.062      0.517      0.605      -0.089       0.153
brand_name_Samsung       -0.0314      0.043     -0.726      0.468      -0.116       0.053
brand_name_Sony          -0.0616      0.050     -1.221      0.222      -0.161       0.037
brand_name_Spice         -0.0148      0.063     -0.234      0.815      -0.139       0.109
brand_name_Vivo          -0.0155      0.048     -0.320      0.749      -0.111       0.080
brand_name_XOLO           0.0151      0.055      0.276      0.783      -0.092       0.123
brand_name_Xiaomi         0.0868      0.048      1.804      0.071      -0.008       0.181
brand_name_ZTE           -0.0058      0.047     -0.122      0.903      -0.099       0.087
os_Others                -0.0519      0.033     -1.585      0.113      -0.116       0.012
os_Windows               -0.0202      0.045     -0.448      0.654      -0.109       0.068
os_iOS                   -0.0669      0.146     -0.457      0.648      -0.354       0.220
4g_yes                    0.0530      0.016      3.341      0.001       0.022       0.084
5g_yes                   -0.0721      0.031     -2.292      0.022      -0.134      -0.010
==============================================================================
Omnibus:                      223.220   Durbin-Watson:                   1.911
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              422.514
Skew:                          -0.618   Prob(JB):                     1.79e-92
Kurtosis:                       4.633   Cond. No.                     1.78e+05
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.78e+05. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Observations:

- adj. R-squared is 0.842, which is good
- the value for const coefficient is 1.3867

Model Performance Check

```
def adjusted_r2(r2, n, p):
    r2 = r2_score(n, p)
    return 1 - ((1 - r2) * (n - 1) / (n - p - 1))


def performance_regression(model, predictor, target):

    pred = model.predict(predictor)

    # R-squared
    r2 = r2_score(target, pred)

    # Adjusted R-squared
    adjr2 = adjusted_r2(predictor, target, pred)
```

```
    # RMSE
    rmse = np.sqrt(mean_squared_error(target, pred))

    # MAE
    mae = mean_absolute_error(target, pred)

    # MAPE
    mape = np.mean(np.abs(target - pred) / target) * 100

    # creating a dataframe of metrics
    dfPerformanceRegression = pd.DataFrame(
        {
            "RMSE": rmse,
            "MAE": mae,
            "R-squared": r2,
            "Adj. R-squared": adjr2,
            "MAPE": mape,
        },
        index=[0],
    )

    return dfPerformanceRegression
```

```
# Model Training Performance
olsmodel1_training = performance_regression(olsmodel1, x_train, y_train)
olsmodel1_training
```

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.229856	0.180302	0.844924	1.528997	4.326213

```
# Model Test Performance
olsmodel1_test = performance_regression(olsmodel1, x_test, y_test)
olsmodel1_test
```

	RMSE	MAE	R-squared	Adj. R-squared	MAPE
0	0.238482	0.184868	0.842315	NaN	4.505694

Observations

- The training R^2 is 0.84, so the model is not underfitting
- The train and test RMSE and MAE are comparable, so the model is not overfitting either
- MAE suggests that the model can predict used price devices within a mean error of 0.18 on the test data
- MAPE of 4.5 on the test data means that we are able to predict within 4.5% of the used price devices

Checking Linear Regression Assumptions

We will be checking the following Linear Regression assumptions:

- No Multicollinearity
- Linearity of variables
- Independence of error terms
- Normality of error terms
- No Heteroscedasticity

1. No Multicollinearity

- We will test for multicollinearity using VIF.
- **General Rule of thumb**:
  - If VIF is between 1 and 5, then there is low multicollinearity.
  - If VIF is between 5 and 10, we say there is moderate multicollinearity.
  - If VIF is exceeding 10, it shows signs of high multicollinearity.

Let's define a function to check VIF

```
# Function to check VIF.

def checking_vif(predictors):
    vif = pd.DataFrame()
    vif["feature"] = predictors.columns

    vif["VIF"] = [
        variance_inflation_factor(predictors.values, i)
        for i in range(len(predictors.columns))
    ]
    return vif
```

```
checking_vif(x_train)
```

	feature	VIF
0	const	260.828671
1	screen_size	7.677118
2	main_camera_mp	2.284685
3	selfie_camera_mp	2.810716
4	int_memory	1.364057
5	ram	2.255246
6	battery	4.081715
7	weight	6.397183
8	days_used	2.660002
9	normalized_new_price	3.119569
10	release_age	4.898360
11	brand_name_Alcatel	3.405674
12	brand_name_Apple	13.054832
13	brand_name_Asus	3.332003
14	brand_name_BlackBerry	1.632250
15	brand_name_Celkon	1.774820
16	brand_name_Coolpad	1.467981
17	brand_name_Gionee	1.951247
18	brand_name_Google	1.321771
19	brand_name_HTC	3.410252
20	brand_name_Honor	3.340621
21	brand_name_Huawei	5.983857
22	brand_name_Infinix	1.283814
23	brand_name_Karbonn	1.573683
24	brand_name_LG	4.849589
25	brand_name_Lava	1.711317
26	brand_name_Lenovo	4.558847
27	brand_name_Meizu	2.179607
28	brand_name_Micromax	3.363518
29	brand_name_Microsoft	1.869558
30	brand_name_Motorola	3.274455
31	brand_name_Nokia	3.473140
32	brand_name_OnePlus	1.437047
33	brand_name_Oppo	3.971065
34	brand_name_Others	9.710921
35	brand_name_Panasonic	2.105711
36	brand_name_Realme	1.946675
37	brand_name_Samsung	7.539832
38	brand_name_Sony	2.943127
39	brand_name_Spice	1.688868
40	brand_name_Vivo	3.651320
41	brand_name_XOLO	2.138074
42	brand_name_Xiaomi	3.719678
43	brand_name_ZTE	3.797527
44	os_Others	1.854134
45	os_Windows	1.595291
46	os_iOS	11.780766
47	4g_yes	2.468374
48	5g_yes	1.811042

Observations:

- There are few columns with very high VIF values, indicating presence of strong multicollinearity
- We will systematically drop numerical columns with VIF > 5
- We will ignore the VIF values for dummy variables and the constant (intercept)

Removing multicollinearity

```
def validate_multicollinearity(predictors, target, columns):
    adj_r2 = []
    rmse = []

    for column in columns:
        train = predictors.loc[:, ~predictors.columns.str.startswith(column)]

        # create the model
        olsmodel = sm.OLS(target, train).fit()

        adj_r2.append(olsmodel.rsquared_adj)
```

```
        rmse.append(np.sqrt(olsmodel.mse_resid))

    # creating new dataframe
    tempDataframe = pd.DataFrame(
        {
            "col": columns,
            "Adj. R-squared": adj_r2,
            "RMSE": rmse,
        }
    ).sort_values(by="Adj. R-squared", ascending=False)
    tempDataframe.reset_index(drop=True, inplace=True)

    return tempDataframe
```

```
cols = ["screen_size","weight","brand_name_Apple","brand_name_Huawei","brand_name_Others","brand_name_Samsung","os_iOS"]

result = validate_multicollinearity(x_train, y_train, cols)
result
```

	col	Adj. R-squared	RMSE
0	brand_name_Apple	0.841847	0.232173
1	brand_name_Huawei	0.841847	0.232173
2	brand_name_Others	0.841844	0.232175
3	os_iOS	0.841833	0.232183
4	brand_name_Samsung	0.841812	0.232199
5	screen_size	0.838427	0.234670
6	weight	0.838102	0.234906

Next steps: Generate code with result | View recommended plots | New interactive sheet

```
col_to_drop = 'brand_name_Apple'
x_train2 = x_train.loc[:, ~x_train.columns.str.startswith(col_to_drop)]
x_test2 = x_test.loc[:, ~x_test.columns.str.startswith(col_to_drop)]

# Check VIF after drop column
vif = checking_vif(x_train2)
print("VIF after dropping", col_to_drop)
vif
```

VIF after dropping brand_name_Apple

	feature	VIF
0	const	260.140559
1	screen_size	7.643504
2	main_camera_mp	2.284075
3	selfie_camera_mp	2.789591
4	int_memory	1.364046
5	ram	2.247171
6	battery	4.079641
7	weight	6.394451
8	days_used	2.659521
9	normalized_new_price	3.102357
10	release_age	4.889730
11	brand_name_Alcatel	3.230621
12	brand_name_Asus	3.144923
13	brand_name_BlackBerry	1.561068
14	brand_name_Celkon	1.731859
15	brand_name_Coolpad	1.436792
16	brand_name_Gionee	1.886305
17	brand_name_Google	1.293140
18	brand_name_HTC	3.240385
19	brand_name_Honor	3.159809
20	brand_name_Huawei	5.581499
21	brand_name_Infinix	1.265388
22	brand_name_Karbonn	1.544215
23	brand_name_LG	4.564980
24	brand_name_Lava	1.670716
25	brand_name_Lenovo	4.291535
26	brand_name_Meizu	2.092794
27	brand_name_Micromax	3.214160
28	brand_name_Microsoft	1.835387
29	brand_name_Motorola	3.109598
30	brand_name_Nokia	3.258328
31	brand_name_OnePlus	1.402472
32	brand_name_Oppo	3.762609
33	brand_name_Others	9.075017
34	brand_name_Panasonic	2.030879
35	brand_name_Realme	1.878667
36	brand_name_Samsung	6.991243
37	brand_name_Sony	2.799041
38	brand_name_Spice	1.655390
39	brand_name_Vivo	3.447542
40	brand_name_XOLO	2.069721
41	brand_name_Xiaomi	3.512985
42	brand_name_ZTE	3.604237
43	os_Others	1.734615
44	os_Windows	1.593031
45	os_iOS	1.908364
46	4g_yes	2.467374
47	5g_yes	1.802632

Next steps: Generate code with vif | View recommended plots | New interactive sheet

```
col_to_drop = 'brand_name_Huawei'
x_train2 = x_train2.loc[:, ~x_train2.columns.str.startswith(col_to_drop)]
x_test2 = x_test2.loc[:, ~x_test2.columns.str.startswith(col_to_drop)]

# Check VIF after drop column
vif = checking_vif(x_train2)
print("VIF after dropping", col_to_drop)
vif
```

VIF after dropping brand_name_Huawei

	feature	VIF
0	const	200.637490
1	screen_size	7.631104
2	main_camera_mp	2.283711
3	selfie_camera_mp	2.779470
4	int_memory	1.362385
5	ram	2.246349
6	battery	4.079120
7	weight	6.393668
8	days_used	2.659043
9	normalized_new_price	3.102052
10	release_age	4.889645
11	brand_name_Alcatel	1.434102
12	brand_name_Asus	1.367353
13	brand_name_BlackBerry	1.158162
14	brand_name_Celkon	1.257613
15	brand_name_Coolpad	1.080724
16	brand_name_Gionee	1.169456
17	brand_name_Google	1.059215
18	brand_name_HTC	1.400689
19	brand_name_Honor	1.356389
20	brand_name_Infinix	1.070622
21	brand_name_Karbonn	1.137125
22	brand_name_LG	1.611735
23	brand_name_Lava	1.145094
24	brand_name_Lenovo	1.563533
25	brand_name_Meizu	1.184730
26	brand_name_Micromax	1.480613
27	brand_name_Microsoft	1.540355
28	brand_name_Motorola	1.381647
29	brand_name_Nokia	1.707052
30	brand_name_OnePlus	1.084218
31	brand_name_Oppo	1.461789
32	brand_name_Others	2.440171
33	brand_name_Panasonic	1.188347
34	brand_name_Realme	1.196206
35	brand_name_Samsung	2.000057
36	brand_name_Sony	1.345155
37	brand_name_Spice	1.163398
38	brand_name_Vivo	1.399635
39	brand_name_XOLO	1.240836
40	brand_name_Xiaomi	1.412064
41	brand_name_ZTE	1.453054
42	os_Others	1.728109
43	os_Windows	1.592826
44	os_iOS	1.223941
45	4g_yes	2.447686
46	5g_yes	1.801264

Next steps: Generate code with vif | View recommended plots | New interactive sheet

```
col_to_drop = 'screen_size'
x_train2 = x_train2.loc[:, ~x_train2.columns.str.startswith(col_to_drop)]
x_test2 = x_test2.loc[:, ~x_test2.columns.str.startswith(col_to_drop)]

# Check VIF after drop column
vif = checking_vif(x_train2)
print("VIF after dropping", col_to_drop)
vif
```

VIF after dropping screen_size

	feature	VIF
0	const	170.582030
1	main_camera_mp	2.280356
2	selfie_camera_mp	2.777471
3	int_memory	1.360180
4	ram	2.246286
5	battery	3.836841
6	weight	2.986796
7	days_used	2.647531
8	normalized_new_price	3.056412
9	release_age	4.715114
10	brand_name_Alcatel	1.432579
11	brand_name_Asus	1.365028
12	brand_name_BlackBerry	1.157330
13	brand_name_Celkon	1.257613
14	brand_name_Coolpad	1.080702
15	brand_name_Gionee	1.162697
16	brand_name_Google	1.057510
17	brand_name_HTC	1.395096
18	brand_name_Honor	1.354431
19	brand_name_Infinix	1.070481
20	brand_name_Karbonn	1.136295
21	brand_name_LG	1.602159
22	brand_name_Lava	1.145093
23	brand_name_Lenovo	1.561960
24	brand_name_Meizu	1.183306
25	brand_name_Micromax	1.478784
26	brand_name_Microsoft	1.539498
27	brand_name_Motorola	1.374476
28	brand_name_Nokia	1.696276
29	brand_name_OnePlus	1.084194
30	brand_name_Oppo	1.459752
31	brand_name_Others	2.403494
32	brand_name_Panasonic	1.188239
33	brand_name_Realme	1.194722
34	brand_name_Samsung	1.992368
35	brand_name_Sony	1.342363
36	brand_name_Spice	1.159952
37	brand_name_Vivo	1.399634
38	brand_name_XOLO	1.240818
39	brand_name_Xiaomi	1.409810
40	brand_name_ZTE	1.449074
41	os_Others	1.517179
42	os_Windows	1.592674
43	os_iOS	1.216920
44	4g_yes	2.446714
45	5g_yes	1.796517

Next steps: Generate code with vif | View recommended plots | New interactive sheet

Dropping high p-value variables

Observations: we will drop the predictor variables having a p-value greater than 0.05 as they do not significantly impact the target variable

```
predictors = x_train2.copy()
cols = predictors.columns.tolist()

# setting an initial max p-value
max_p_value = 1

while len(cols) > 0:
    # defining the train set
    x_train_aux = predictors[cols]

    # fitting the model
    model = sm.OLS(y_train, x_train_aux).fit()

    # getting the p-values and the maximum p-value
    p_values = model.pvalues
    max_p_value = max(p_values)
```